

Lenguajes de Consulta para XML

Marcelo Arenas
P. Universidad Católica de Chile

Indice

- Lenguajes de consulta para XML.
- Core XPath.
- Lógica de primer orden y XML.
- Conditional XPath.
- Regular XPath.
- Lógica de segundo orden monádica y XML.
- Monadic Datalog.
- Consultas con un número arbitrario de argumentos.

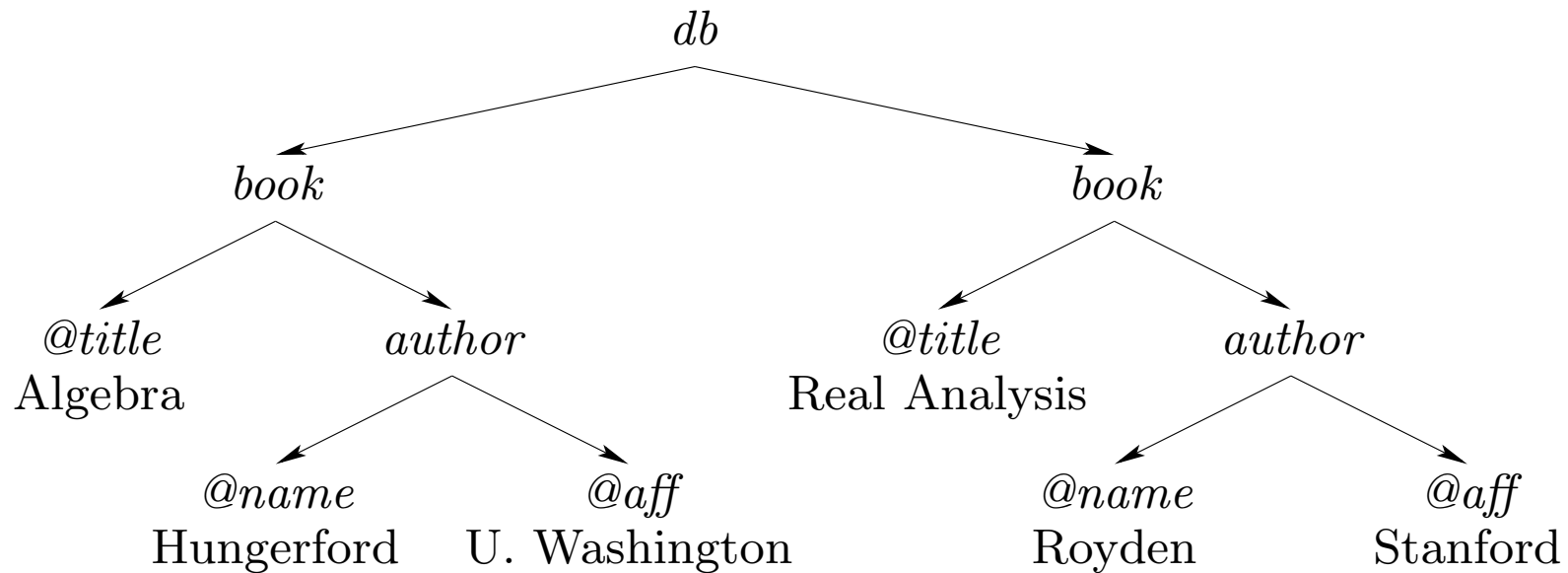
Indice

- Lenguajes de consulta para XML.
- Core XPath.
- Lógica de primer orden y XML.
- Conditional XPath.
- Regular XPath.
- Lógica de segundo orden monádica y XML.
- Monadic Datalog.
- Consultas con un número arbitrario de argumentos.

Documentos XML

```
<db>
  <book title="Algebra">
    <author name="Hungerford" aff="U. Washington">
    </author>
  </book>
  <book title="Real Analysis">
    <author name="Royden" aff="Stanford">
    </author>
  </book>
</db>
```

Documentos XML como árboles



Lenguajes de consulta para XML: Operaciones básicas

Filtrar: Seleccionar valores desde un documento XML.

- Navegación, selección, extracción.

Mezclar: Integrar valores desde múltiples fuentes.

- Join, agregación.

Transformar valores desde un esquema a otro.

- Construcción de documentos XML.

Lenguajes de consulta para XML: Algunos ejemplos

XPath:

- Lenguaje más popular para navegar, seleccionar y extraer valores desde documentos XML.
- Parte de lenguajes más complejos como XQuery y XSLT.

XQuery:

- Genera documentos XML como respuesta.
- Incluye join y agregación.

XSLT:

- Lenguaje de patrones.
- Puede generar como respuesta documentos XML, HTML, texto u otros formatos.

Lenguajes de consulta para XML: Algunos ejemplos

XPath:

- Lenguaje más popular para navegar, seleccionar y extraer valores desde documentos XML.
- Parte de lenguajes más complejos como XQuery y XSLT.

XQuery:

- Genera documentos XML como respuesta.
- Incluye join y agregación.

XSLT:

- Lenguaje de patrones.
- Puede generar como respuesta documentos XML, HTML, texto u otros formatos.

Indice

- Lenguajes de consulta para XML.
- Core XPath.
- Lógica de primer orden y XML.
- Conditional XPath.
- Regular XPath.
- Lógica de segundo orden monádica y XML.
- Monadic Datalog.
- Consultas con un número arbitrario de argumentos.

XML Path Language (XPath)

Estándar de la W3C: <http://www.w3.org/TR/xpath>.

Lenguaje para navegar, seleccionar nodos y extraer valores.

Algunas implementaciones:

- XALAN : Apache Foundation (XSLT)
- XT : James Clark (XSLT)
- SAXON : Michael Kay (XSLT y XQuery)

XML Path Language (XPath)

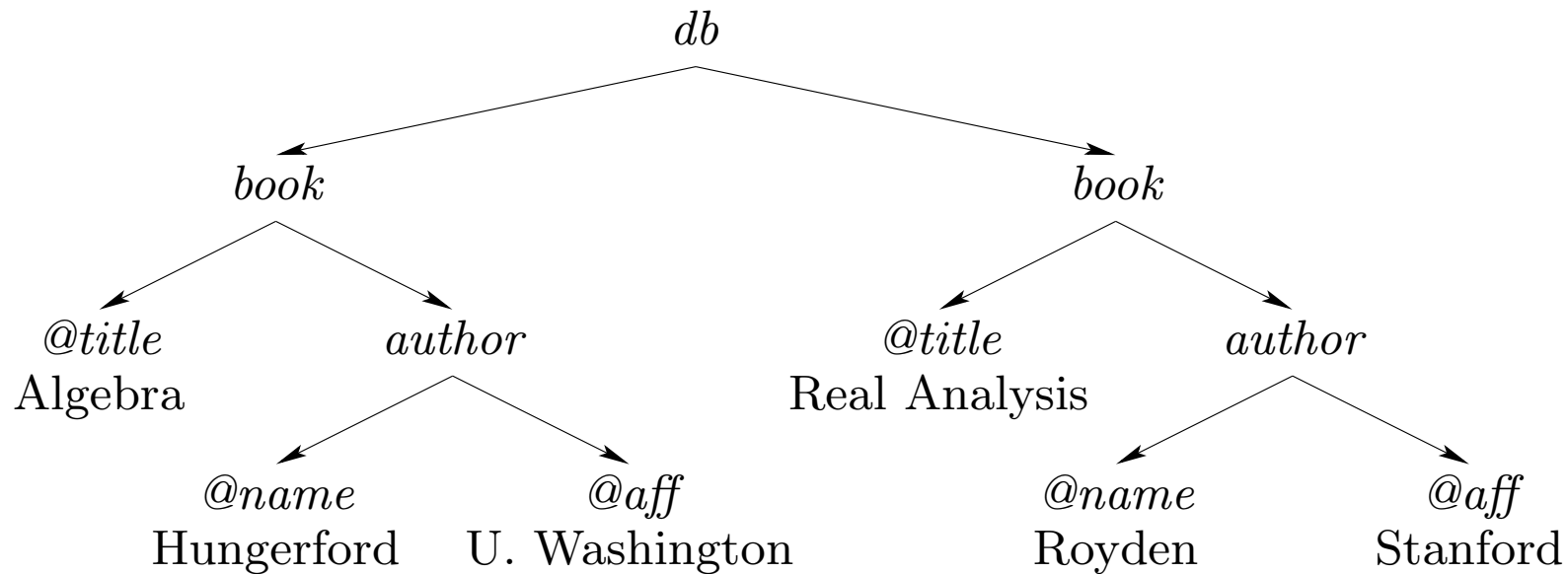
Estándar de la W3C: <http://www.w3.org/TR/xpath>.

Lenguaje para **navegar, seleccionar nodos** y extraer valores.

Algunas implementaciones:

- XALAN : Apache Foundation (XSLT)
- XT : James Clark (XSLT)
- SAXON : Michael Kay (XSLT y XQuery)

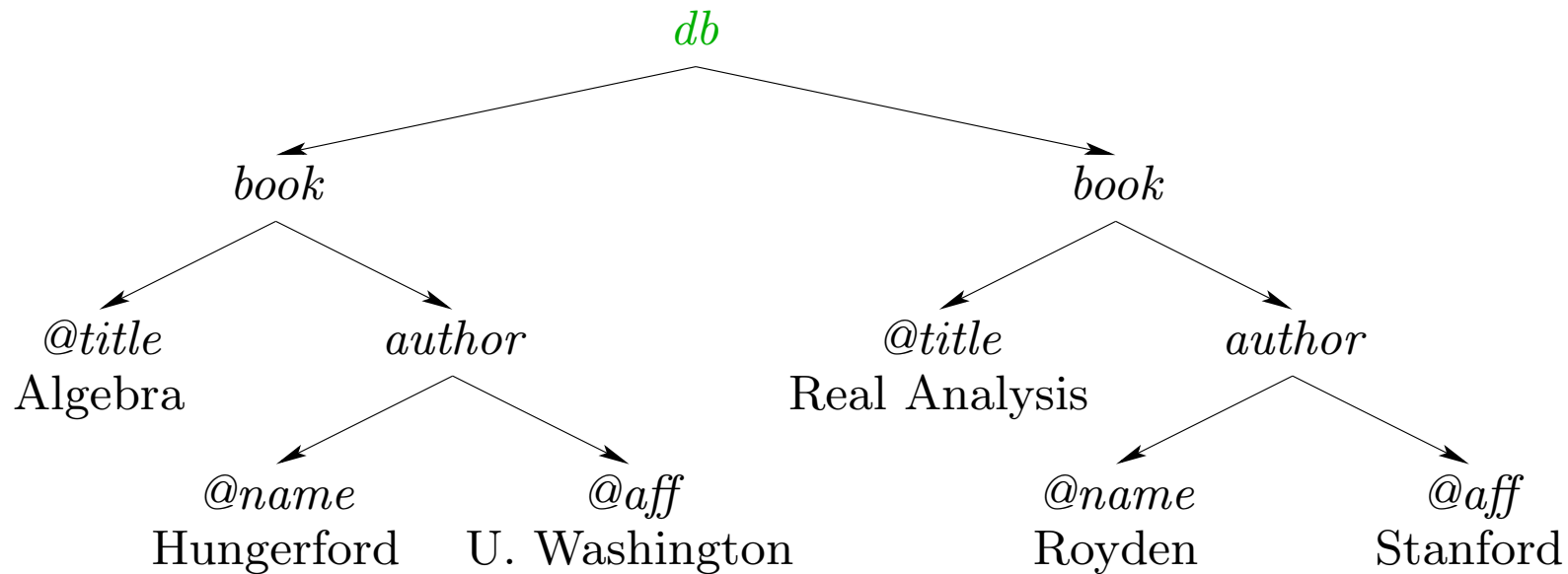
Core XPath: Primer ejemplo



Consulta XPath: *child/?book*

Respuesta:

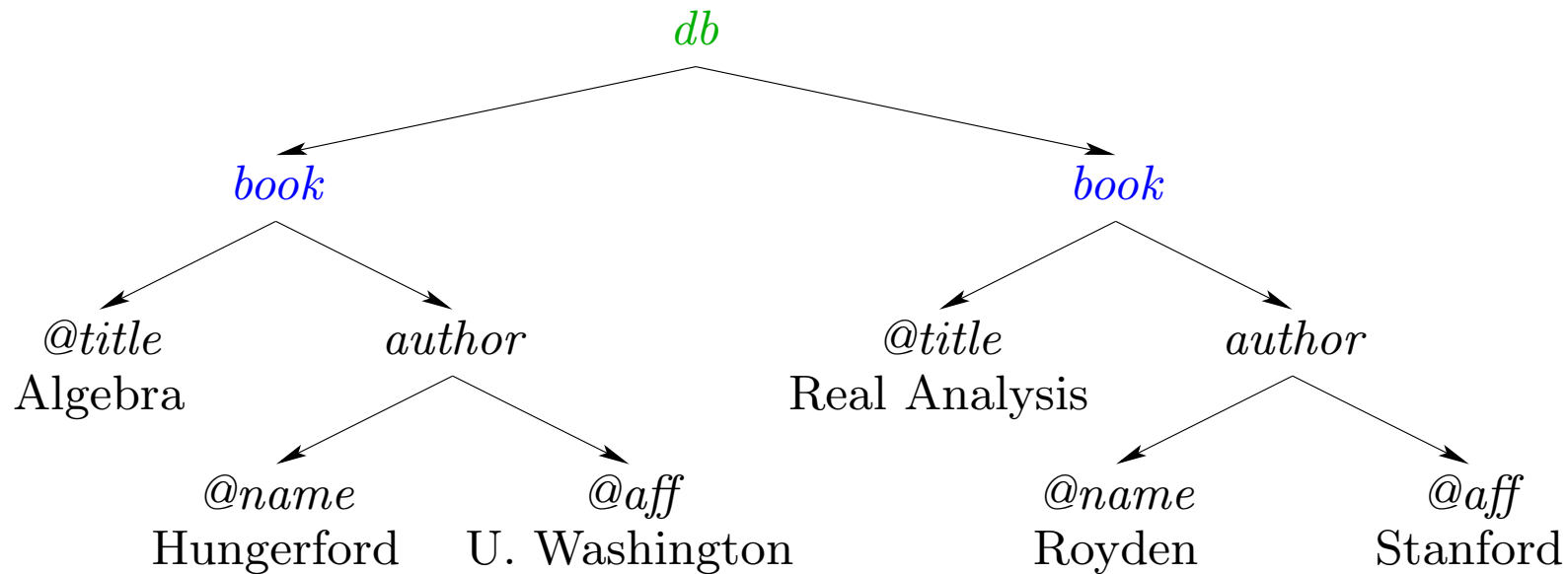
Core XPath: Primer ejemplo



Consulta XPath: *child/?book*

Respuesta:

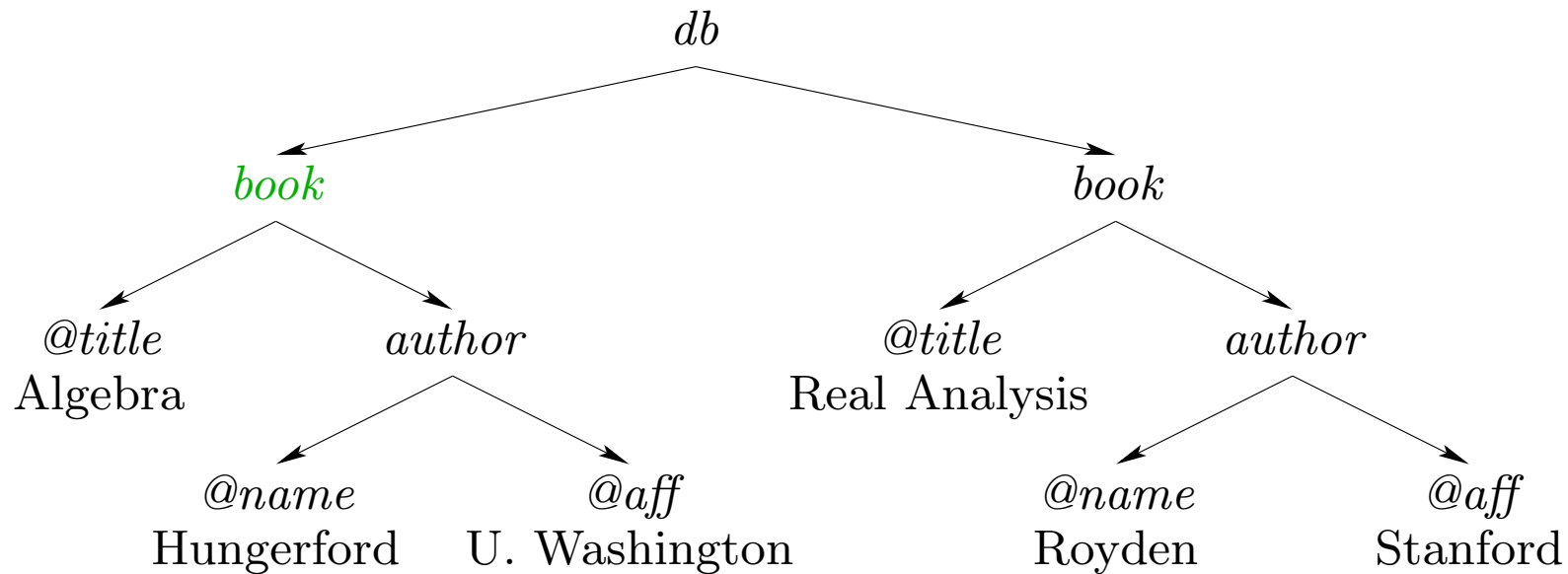
Core XPath: Primer ejemplo



Consulta XPath: `child/?book`

Respuesta: `nodos azules`

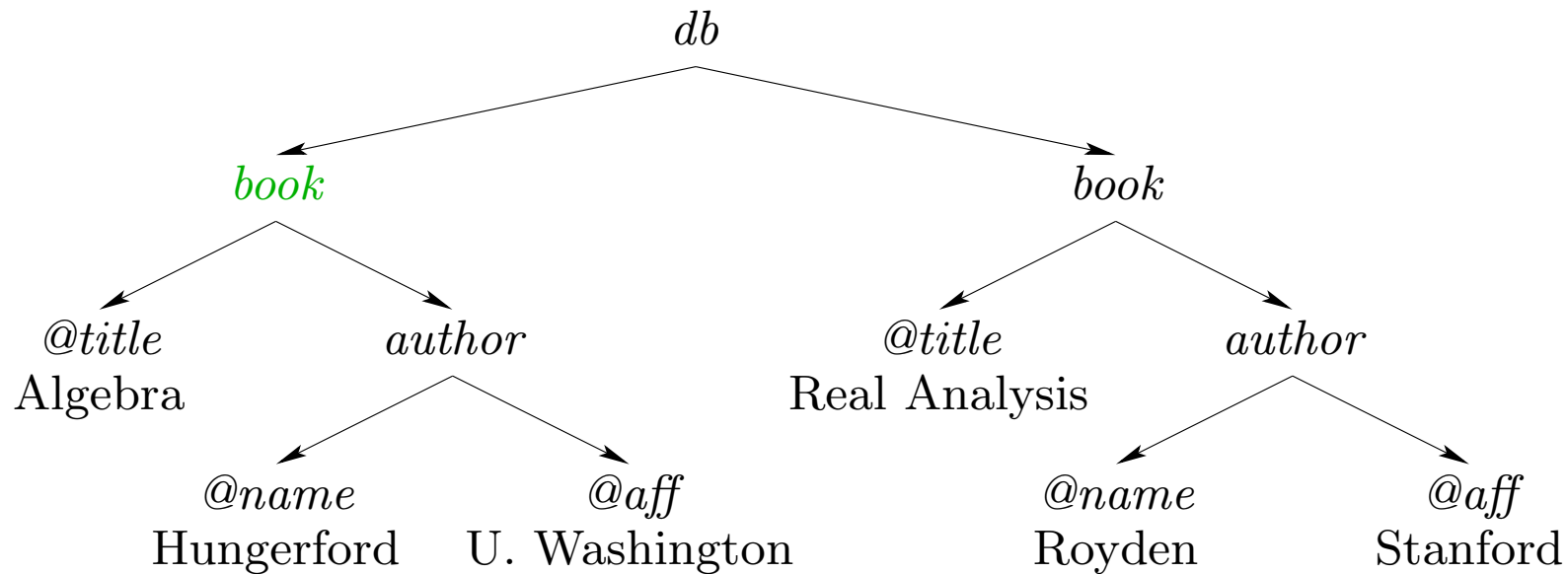
Core XPath: Segundo ejemplo



Consulta XPath: *child/?book*

Respuesta:

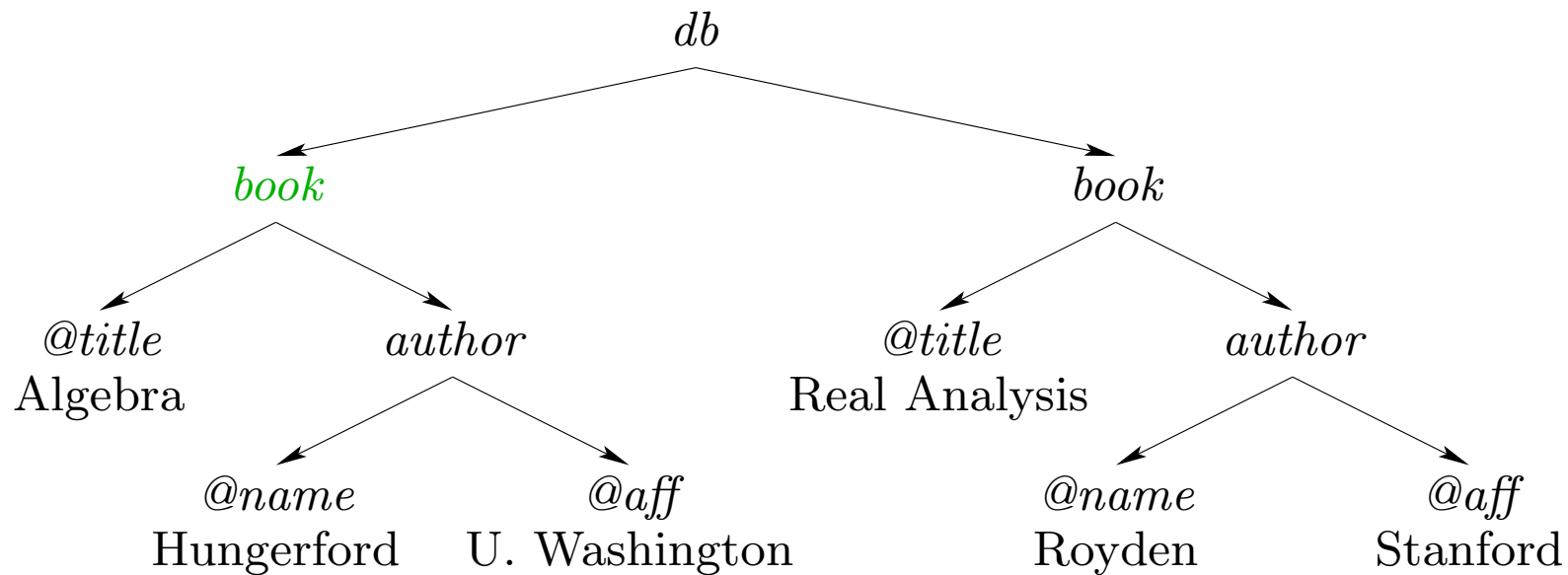
Core XPath: Segundo ejemplo



Consulta XPath: *child/?book*

Respuesta: \emptyset

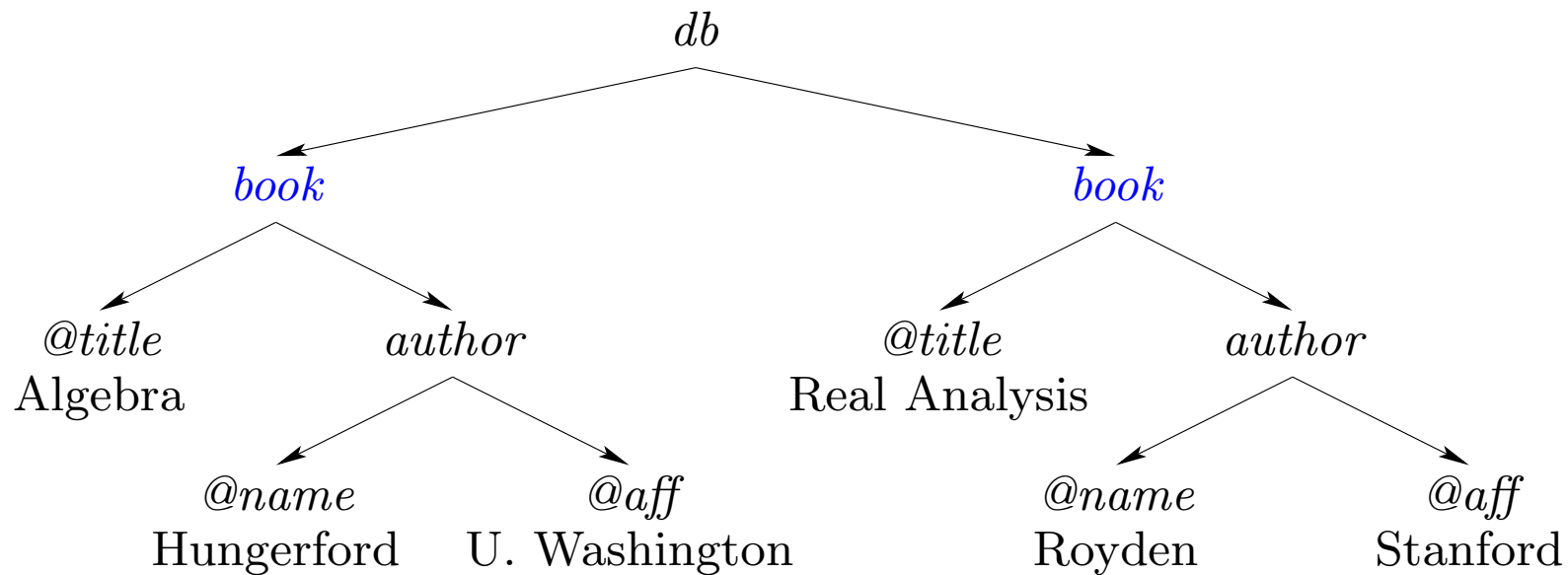
Core XPath: Tercer ejemplo



Consulta XPath: `parent/child/?book`

Respuesta:

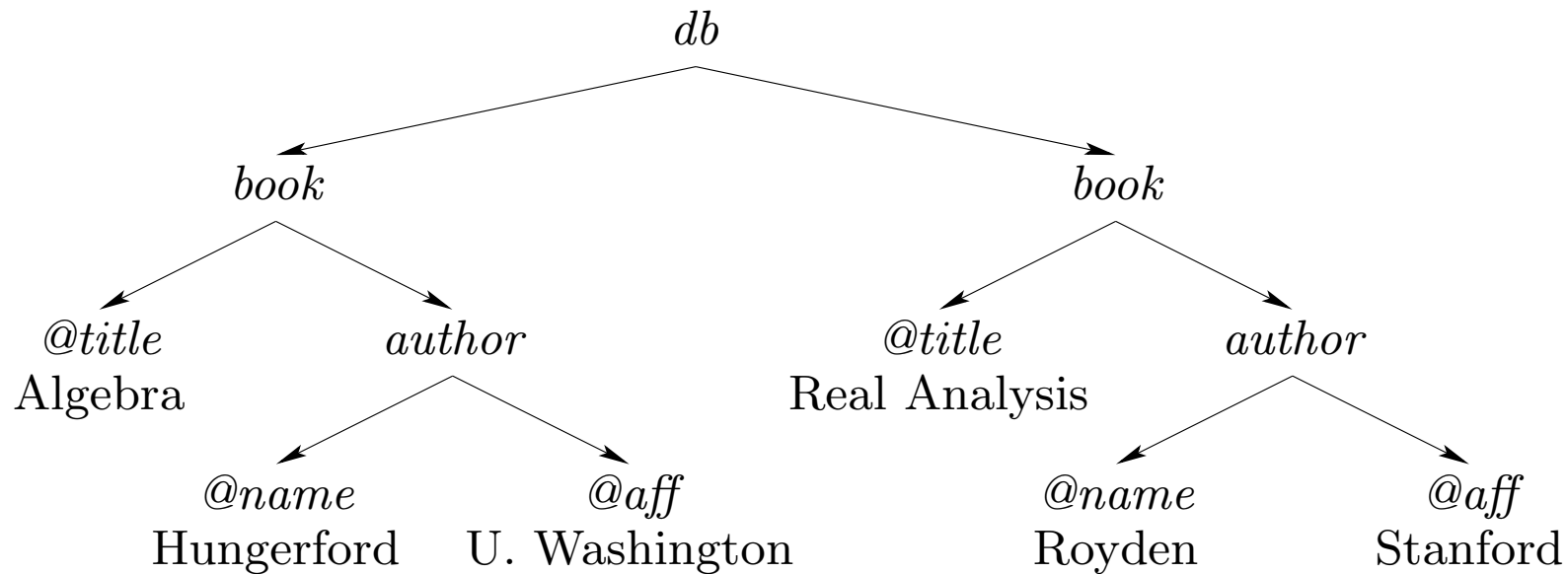
Core XPath: Tercer ejemplo



Consulta XPath: `parent/child/?book`

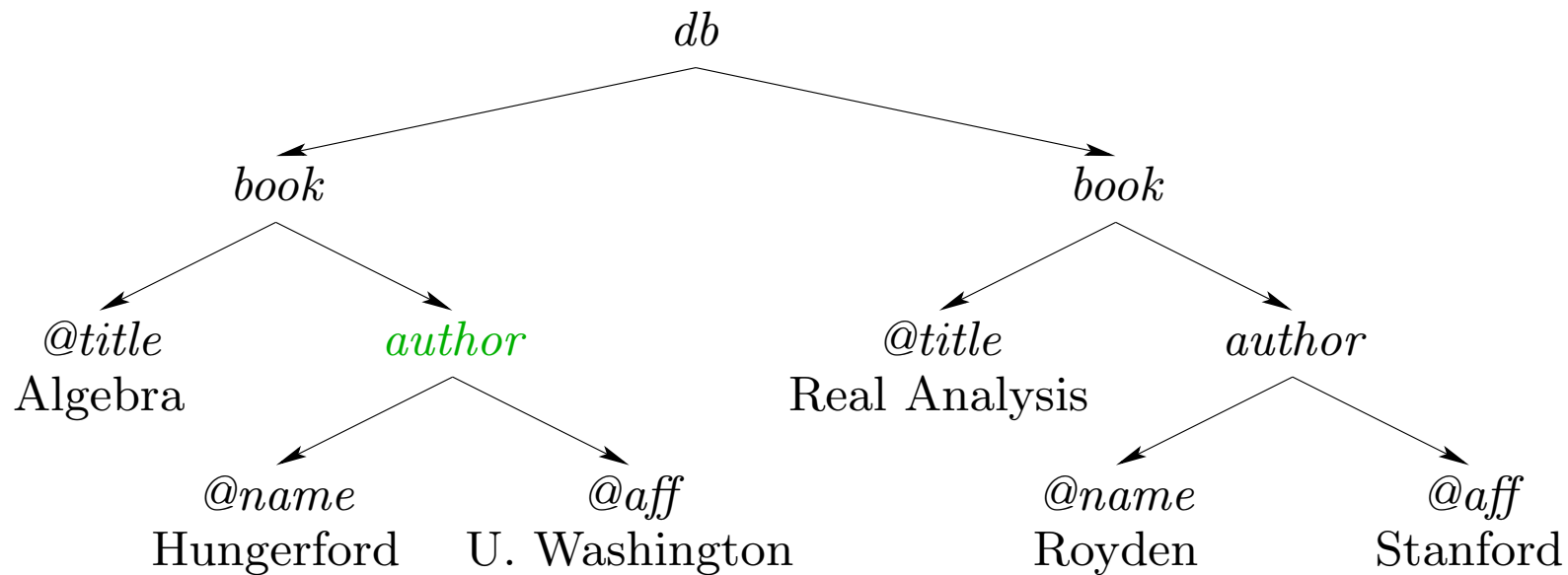
Respuesta: `nodos azules`

Core XPath: Ultimo ejemplo



Consulta XPath: `parent*/child*/?book`

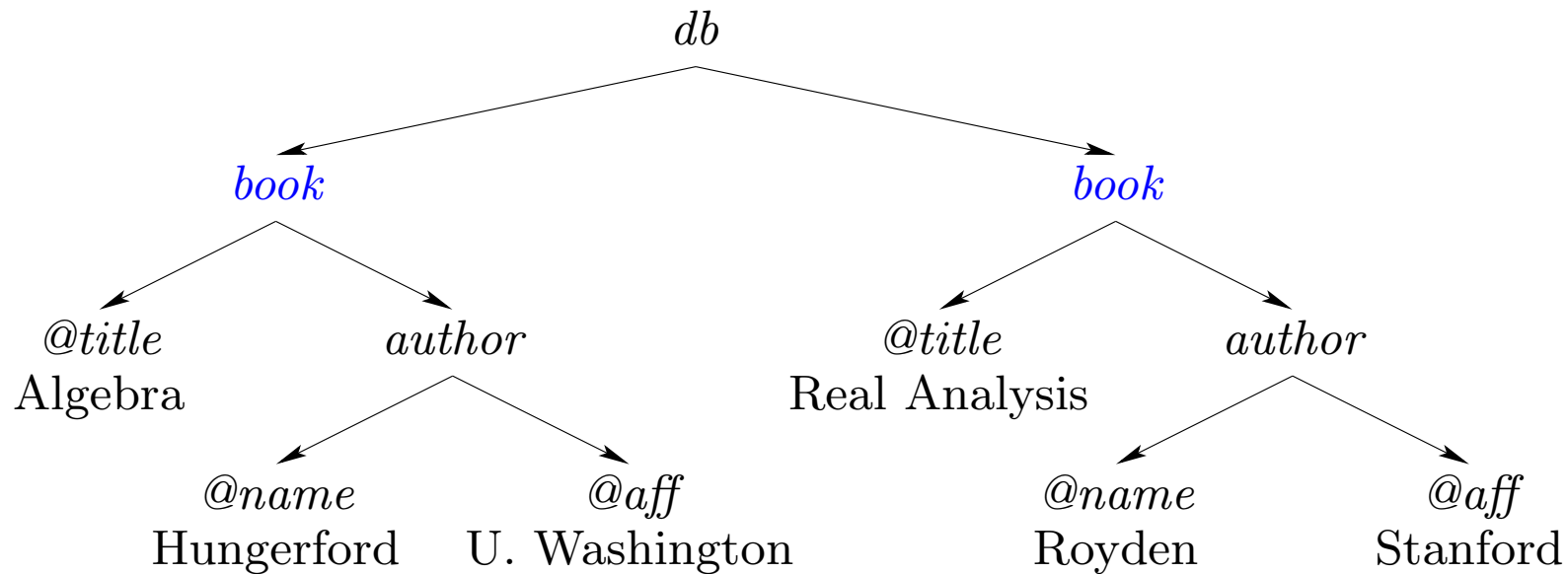
Core XPath: Ultimo ejemplo



Consulta XPath: `parent*/child*/?book`

Respuesta:

Core XPath: Ultimo ejemplo



Consulta XPath: *parent*/child*/?book*

Respuesta: *nodos azules*

Core XPath: Sintaxis

Camino básicos:

`paso ::= child | parent | right | left`

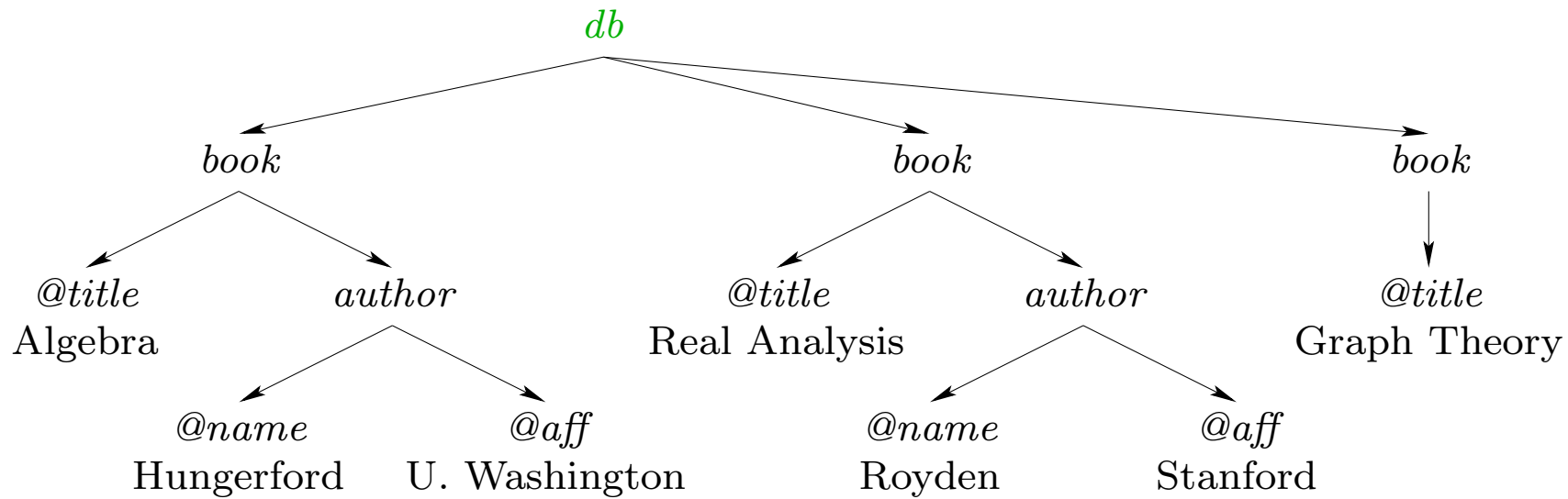
Expresiones para caminos:

`camino ::= paso | paso* | camino/camino |
camino \cup camino | ?test`

Filtros:

`test ::= nombre | <camino> | \neg test | test \wedge test`

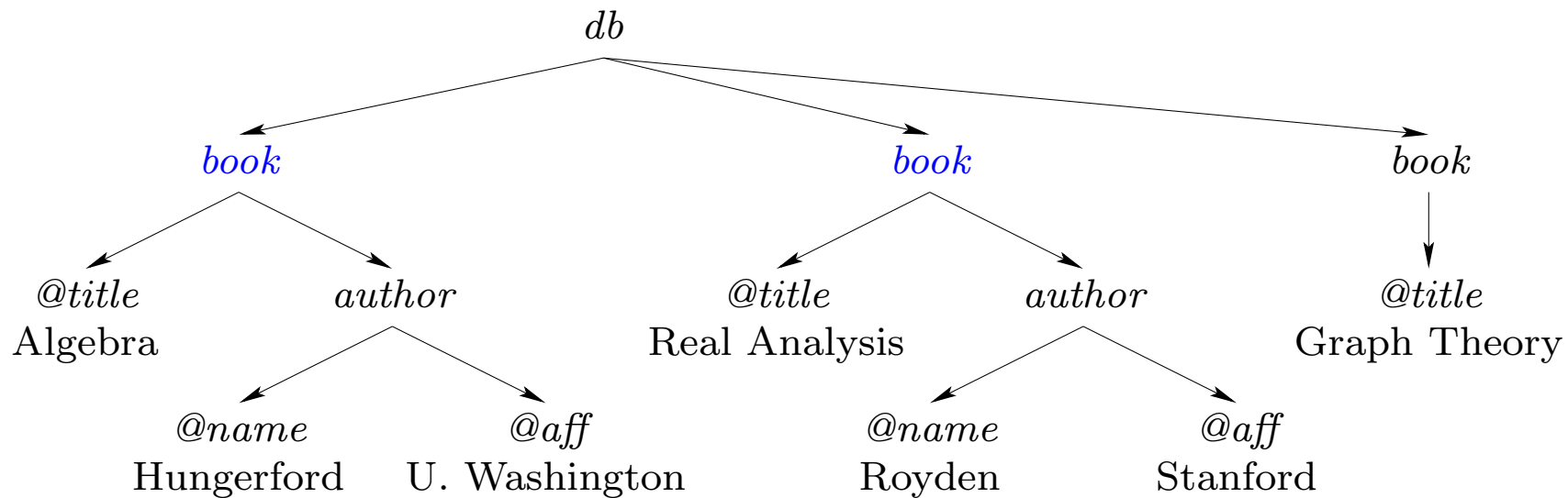
Core XPath: Otro ejemplo



Consulta XPath: `child/?book/?<child/?author>`

Respuesta:

Core XPath: Otro ejemplo



Consulta XPath: `child/?book/?<child/?author>`

Respuesta: **nodos azules**

Core XPath: Semántica para caminos básicos

Dado: Documento XML T .

$$\llbracket \text{child} \rrbracket_T = \{(n_1, n_2) \mid n_2 \text{ es hijo de } n_1 \text{ en } T\}$$

$$\llbracket \text{parent} \rrbracket_T = \llbracket \text{child} \rrbracket_T^{-1}$$

$$\llbracket \text{right} \rrbracket_T = \{(n_1, n_2) \mid \text{existe nodo } n \text{ en } T \text{ tal que } n_1 \text{ es el } i\text{-ésimo hijo de } n \text{ y } n_2 \text{ es el } (i + 1)\text{-ésimo hijo de } n \text{ en } T\}$$

$$\llbracket \text{left} \rrbracket_T = \llbracket \text{right} \rrbracket_T^{-1}$$

Core XPath: Semántica para caminos complejos

$$\begin{aligned} \llbracket \text{camino}_1 / \text{camino}_2 \rrbracket_T &= \llbracket \text{camino}_1 \rrbracket_T \circ \llbracket \text{camino}_2 \rrbracket_T \\ \llbracket \text{camino}_1 \cup \text{camino}_2 \rrbracket_T &= \llbracket \text{camino}_1 \rrbracket_T \cup \llbracket \text{camino}_2 \rrbracket_T \\ \llbracket \text{paso}^* \rrbracket_T &= \{(n, n) \mid n \text{ está en } T\} \cup \llbracket \text{paso} \rrbracket_T \cup \\ &\quad \llbracket \text{paso/paso} \rrbracket_T \cup \llbracket \text{paso/paso/paso} \rrbracket_T \cup \dots \\ \llbracket \text{?test} \rrbracket_T &= \{(n, n) \mid n \in \llbracket \text{test} \rrbracket_T\} \end{aligned}$$

Core XPath: Semántica para filtros

$$\llbracket \text{nombre} \rrbracket_T = \{n \text{ en } T \mid \text{la etiqueta de } n \text{ en } T \text{ es } \text{nombre}\}$$

$$\llbracket \langle \text{camino} \rangle \rrbracket_T = \{n \text{ en } T \mid \text{existe } n' \text{ en } T \text{ tal que } (n, n') \in \llbracket \text{camino} \rrbracket_T\}$$

$$\llbracket \neg \text{test} \rrbracket_T = \{n \text{ en } T \mid n \notin \llbracket \text{test} \rrbracket_T\}$$

$$\llbracket \text{test}_1 \wedge \text{test}_2 \rrbracket_T = \llbracket \text{test}_1 \rrbracket_T \cap \llbracket \text{test}_2 \rrbracket_T$$

¿Cuándo es *bueno* un lenguaje de consulta?

Criterios esenciales:

- Expresividad.
- Complejidad.

¡Estos objetivos se contraponen!

- Vamos a ver que pasa en el caso de Core XPath.

Core XPath: Complejidad

Enfoque ingenuo para evaluar una consulta: Usar un algoritmo recursivo que procesa secuencialmente la consulta.

- Usado en la mayoría de las implementaciones de XPath.

En XALAN y XT: Si $Q = p_1/p_2/\dots/p_k$ ($k \geq 1$), donde cada p_i es un paso o un test, entonces Q es procesada de la siguiente forma.

procesar(Q : consulta, T : árbol, n : nodo)

$N := \{n' \mid (n, n') \in \llbracket p_1 \rrbracket_T\}$

if $k = 1$ **then return** N

else

$R := \emptyset$

for each $n' \in N$ **do** $R := R \cup \text{procesar}(p_2/\dots/p_k, T, n')$

return R

Core XPath: Complejidad

¡Enfoque ingenuo es exponencial!

- Incluso si consideramos documentos de tamaño fijo.

Ejemplo [GKP05]: Considere el documento usado antes y la siguiente secuencia de consultas.

$Q_1 = ?db/child/?book$

$Q_2 = ?db/child/?book/parent/?db/child/?book$

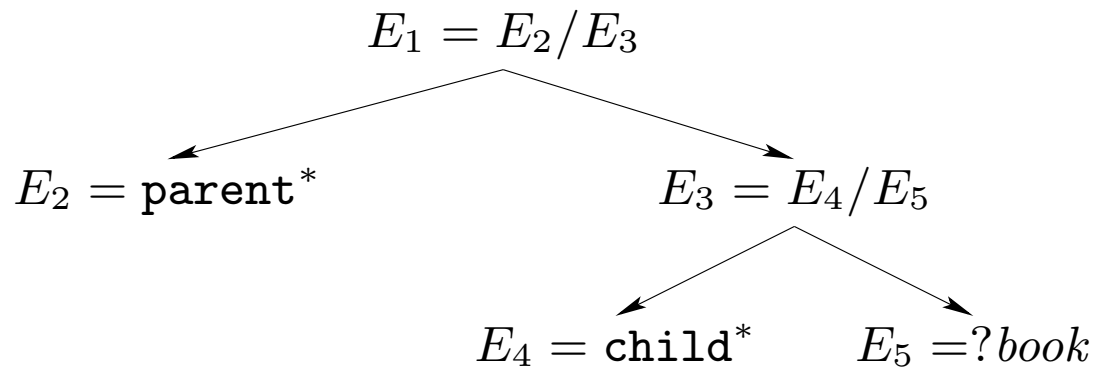
$Q_3 = ?db/child/?book/parent/?db/child/?book/parent/?db/child/?book$

...

Core XPath: Evaluación bottom-up

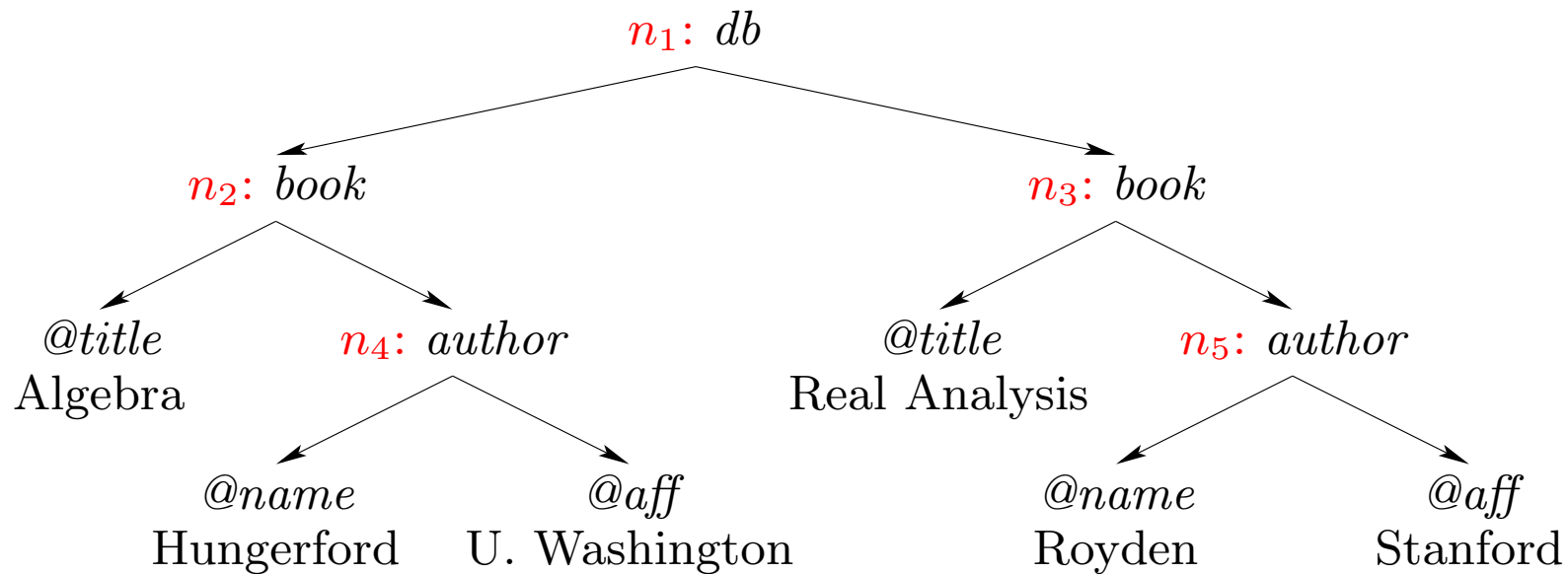
Algoritmo Bottom-up [GKP05]: $\text{parent}^*/\text{child}^*/?book$

Primer paso: Construir el árbol de parsing de la consulta.



Core XPath: Evaluación bottom-up

Segundo paso: Evaluar las sub-consultas de manera bottom-up.



Core XPath: Evaluación bottom-up

$[[E_4]]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_1	n_2
n_1	n_3
n_1	n_4
n_1	n_5
n_2	n_4
n_3	n_5

Core XPath: Evaluación bottom-up

$\llbracket E_4 \rrbracket_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_1	n_2
n_1	n_3
n_1	n_4
n_1	n_5
n_2	n_4
n_3	n_5

$\llbracket E_5 \rrbracket_T$	
n_2	n_2
n_3	n_3

Core XPath: Evaluación bottom-up

$[[E_4]]_T$		$[[E_5]]_T$		$[[E_3]]_T$	
n_1	n_1	n_2	n_2	n_2	n_2
n_2	n_2	n_3	n_3	n_3	n_3
n_3	n_3			n_1	n_2
n_4	n_4			n_1	n_3
n_5	n_5				
n_1	n_2				
n_1	n_3				
n_1	n_4				
n_1	n_5				
n_2	n_4				
n_3	n_5				

Core XPath: Evaluación bottom-up

$[[E_2]]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_2	n_1
n_4	n_2
n_4	n_1
n_3	n_1
n_5	n_3
n_5	n_1

Core XPath: Evaluación bottom-up

$\llbracket E_2 \rrbracket_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_2	n_1
n_4	n_2
n_4	n_1
n_3	n_1
n_5	n_3
n_5	n_1

$\llbracket E_3 \rrbracket_T$	
n_2	n_2
n_3	n_3
n_1	n_2
n_1	n_3

Core XPath: Evaluación bottom-up

$\llbracket E_2 \rrbracket_T$		$\llbracket E_3 \rrbracket_T$		$\llbracket E_1 \rrbracket_T$	
n_1	n_1	n_2	n_2	n_1	n_2
n_2	n_2	n_3	n_3	n_1	n_3
n_3	n_3	n_1	n_2	n_2	n_2
n_4	n_4	n_1	n_3	n_2	n_3
n_5	n_5			n_3	n_2
n_2	n_1			n_3	n_3
n_4	n_2			n_4	n_2
n_4	n_1			n_4	n_3
n_3	n_1			n_5	n_2
n_5	n_3			n_5	n_3
n_5	n_1				

Core XPath: Evaluación bottom-up

Teorema [GKP05]: Una Consulta Q en Core XPath puede ser evaluada en tiempo $O(|T| \cdot |Q|)$.

Nota: Dado un nodo n , es posible calcular $\{n' \mid (n, n') \in \llbracket Q \rrbracket_T\}$ en tiempo $O(|T| \cdot |Q|)$.

El algoritmo puede ser extendido para XPath, pero el tiempo de evaluación crece a $O(|T|^4 \cdot |Q|^2)$.

Sabemos que en términos de complejidad Core XPath es un buen lenguaje. ¿Es también bueno en términos de expresividad?

¿Qué tan expresivo es Core XPath?

La respuesta es relativa.

- Es *más, tan o menos* expresivo que ...

¿Con qué lenguaje podemos comparar?

- Un poco de historia: Bases de datos relacionales.

Indice

- Lenguajes de consulta para XML.
- Core XPath.
- **Lógica de primer orden y XML.**
- Conditional XPath.
- Regular XPath.
- Lógica de segundo orden monádica y XML.
- Monadic Datalog.
- Consultas con un número arbitrario de argumentos.

Bases de datos relacionales: Lógica de primer orden

Base de datos relacional: Información es almacenada en relaciones (tablas).

Lenguaje de consulta natural: **Lógica de primer orden (FO)**.

- Ha sido estudiada por más de 100 años.
- Sintaxis y semántica bien definida.
- Expresividad bien entendida: que se puede y que no se puede decir.
- Complejidad bien entendida.

Pero existe un problema: Difícil de optimizar.

Bases de datos relacionales: Algebra relacional

Un segundo lenguaje de consulta: **Algebra Relacional**.

- Combinación de operaciones algebraicas: selección, proyección, join, unión, diferencia, ...

Ventaja: Fácil de implementar y optimizar.

- Una de las razones para el éxito de las bases de datos relacionales.

Pero ¿cuál es la expresividad del álgebra relacional?

Teorema [Cod72]: Algebra relacional = FO.

FO y XML

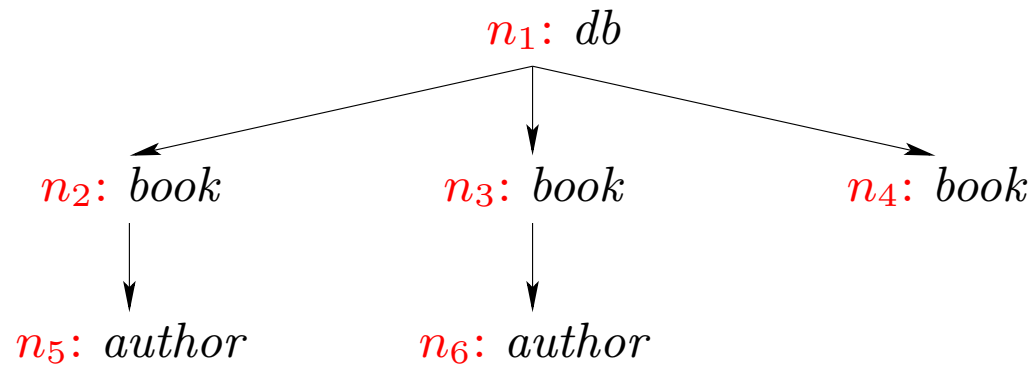
¿Podemos utilizar lógica de primer orden como un lenguaje de consulta para XML?

- ¿Es fácil expresar una consulta?

¿Cuál es la relación entre Core XPath y FO?

FO sobre árboles

Dado árbol T :



Representamos T como una estructura relacional:

$$\mathcal{A}_T = \langle D, \prec_{\text{ch}}, \prec_{\text{ch}}^*, \prec_{\text{sb}}, \prec_{\text{sb}}^*, P_{\text{db}}, P_{\text{book}}, P_{\text{author}} \rangle$$

FO sobre árboles

Donde:

$$D = \{n_1, n_2, n_3, n_4, n_5, n_6\}$$

$$\prec_{\text{ch}} = \{(n_1, n_2), (n_1, n_3), (n_1, n_4), (n_2, n_5), (n_3, n_6)\}$$

$$\prec_{\text{ch}}^* = \{(n_1, n_1), (n_1, n_2), (n_1, n_3), (n_1, n_4), (n_1, n_5), (n_1, n_6), (n_2, n_2), (n_2, n_5), (n_3, n_3), (n_3, n_6), (n_4, n_4), (n_5, n_5), (n_6, n_6)\}$$

$$\prec_{\text{sb}} = \{(n_2, n_3), (n_3, n_4)\}$$

$$\prec_{\text{sb}}^* = \{(n_1, n_1), (n_2, n_2), (n_2, n_3), (n_2, n_4), (n_3, n_3), (n_3, n_4), (n_4, n_4), (n_5, n_5), (n_6, n_6)\}$$

$$P_{db} = \{n_1\}$$

$$P_{book} = \{n_2, n_3, n_4\}$$

$$P_{author} = \{n_5, n_6\}$$

FO sobre árboles: Ejemplos

Veamos como escribir las consultas que formulamos en Core XPath:

`child/?book:`

$$Q_1(x, y) = x \prec_{\text{ch}} y \wedge P_{\text{book}}(y).$$

`parent/child/?book:`

$$Q_2(x, y) = \exists z (z \prec_{\text{ch}} x \wedge z \prec_{\text{ch}} y \wedge P_{\text{book}}(y)).$$

`parent*/child*/?book:`

$$Q_3(x, y) = \exists z (z \prec_{\text{ch}}^* x \wedge z \prec_{\text{ch}}^* y \wedge P_{\text{book}}(y)).$$

`child/?book/?<child/?author>:`

$$Q_4(x, y) = x \prec_{\text{ch}} y \wedge P_{\text{book}}(y) \wedge \exists z (y \prec_{\text{ch}} z \wedge P_{\text{author}}(z)).$$

Core XPath y FO

Core XPath \subseteq FO: Para cada consulta Q en Core XPath existe una fórmula $\varphi(x, y)$ en FO tal que

$$(n, n') \in \llbracket Q \rrbracket_T \quad \text{si y sólo si} \quad \mathfrak{A}_T \models \varphi(n, n').$$

Pero: **FO $\not\subseteq$ Core XPath.**

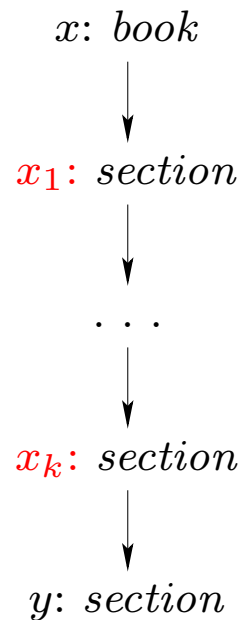
- No podemos expresar algunas consultas naturales.

FO no está contenido en Core XPath

Un ejemplo:

$$Q(x, y) = P_{book}(x) \wedge x \prec_{ch}^* y \wedge P_{section}(y) \wedge \\ \forall z (x \prec_{ch}^* z \wedge x \neq z \wedge z \prec_{ch}^* y \wedge z \neq y \rightarrow P_{section}(z)).$$

Buscamos:



Core XPath

¿Qué tenemos que agregar a Core XPath para alcanzar la expresividad de FO?

¿Es una buena idea agregar nuevos elementos a Core XPath?

- Si lo que vamos a agregar es *natural, útil e implementable eficientemente*, entonces conviene agregarlo.

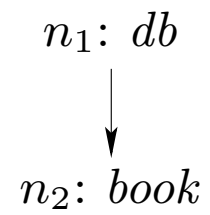
Una pregunta natural: ¿Por qué no usar FO directamente?

Tres desventajas:

- A los usuarios no les gusta trabajar con variables.
- Difícil de optimizar.
- Costoso evaluar una consulta.

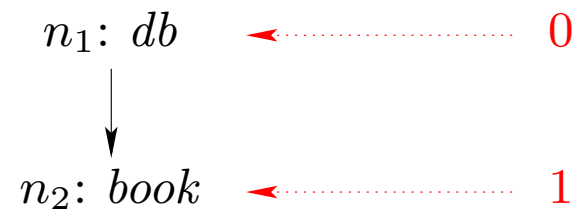
Complejidad de FO sobre árboles

Sea T :



Complejidad de FO sobre árboles

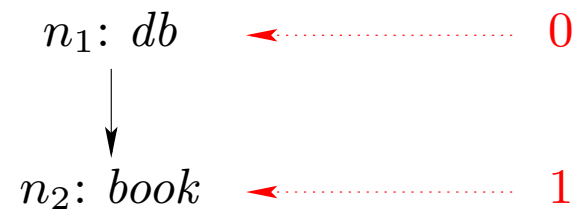
Sea T :



Podemos usar n_1 como 0 y n_2 como 1, y así reducir desde SAT.

Complejidad de FO sobre árboles

Sea T :



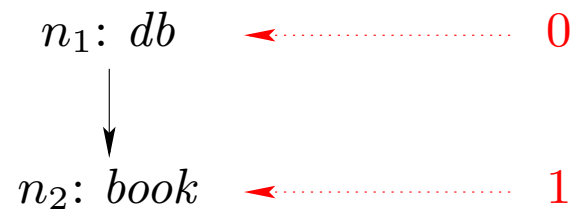
Podemos usar n_1 como 0 y n_2 como 1, y así reducir desde SAT.

Ejemplo: $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z)$ es representado como

$$\exists x \exists y \exists z ((P_{book}(x) \vee P_{db}(y) \vee P_{db}(z)) \wedge (P_{db}(x) \vee P_{book}(y) \vee P_{db}(z))).$$

Complejidad de FO sobre árboles

Sea T :



Podemos usar n_1 como 0 y n_2 como 1, y así reducir desde SAT.

Ejemplo: $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z)$ es representado como

$$\exists x \forall y \exists z ((P_{book}(x) \vee P_{db}(y) \vee P_{db}(z)) \wedge (P_{db}(x) \vee P_{book}(y) \vee P_{db}(z))).$$

Conclusión: FO sobre árboles es **NP-hard**.

Conditional XPath

Buena noticia: Hay que agregar poco a Core XPath para obtener FO y la complejidad no cambia.

Conditional XPath:

- Caminos básicos:

`paso ::= child | parent | right | left`

- Expresiones para caminos:

`camino ::= paso | (paso/?test)* | camino/camino |
camino \cup camino | ?test`

- Filtros:

`test ::= nombre | <camino> | \neg test | test \wedge test`

Conditional XPath

Dado un documento XML T :

$$\begin{aligned} \llbracket (\text{paso}/?\text{test})^* \rrbracket_T &= \{(n, n) \mid n \text{ está en } T\} \cup \llbracket \text{paso}/?\text{test} \rrbracket_T \cup \\ &\quad \llbracket \text{paso}/?\text{test}/\text{paso}/?\text{test} \rrbracket_T \cup \dots \end{aligned}$$

Nótese que Core XPath \subseteq Conditional XPath:

$$\text{paso}^* \text{ es equivalente a } (\text{paso}/?\neg(a \wedge \neg a))^*.$$

Conditional XPath y FO

Teorema [Mar04a, Mar05]: Sobre árboles se tiene que
Conditional XPath = FO.

Teorema [Mar04b]: Una Consulta Q en Conditional XPath
puede ser evaluada en tiempo $O(|T| \cdot |Q|)$.

¡Conditional XPath es un buen lenguaje!

Otra pregunta natural

Si la complejidad de Conditional XPath es $O(|T| \cdot |Q|)$ y FO es NP-hard, ¿Cómo puede ser que Conditional XPath = FO?

Del teorema anterior se puede concluir que **no hay una traducción eficiente de FO a Conditional XPath.**

Del teorema anterior **no** se puede inferir qué lenguaje es mejor para el usuario.

¿Necesitamos más lenguajes de consulta?

Conditional XPath es un buen lenguaje, pero ¿es suficiente?

Para navegar XML es necesario utilizar **expresiones regulares arbitrarias**.

- La mayoría de los lenguajes propuestos las incluyen.

Veamos un lenguaje con expresiones regulares.

Indice

- Lenguajes de consulta para XML.
- Core XPath.
- Lógica de primer orden y XML.
- Conditional XPath.
- **Regular XPath.**
- Lógica de segundo orden monádica y XML.
- Monadic Datalog.
- Consultas con un número arbitrario de argumentos.

Regular XPath: Sintaxis

Camino básicos:

`paso ::= child | parent | right | left`

Expresiones para caminos:

`camino ::= paso | camino* | camino/camino |
camino \cup camino | ?test`

Filtros:

`test ::= nombre | <camino> | \neg test | test \wedge test`

Regular XPath: Semántica

Dado un documento XML T :

$$\begin{aligned} \llbracket \text{camino}^* \rrbracket_T &= \{(n, n) \mid n \text{ está en } T\} \cup \llbracket \text{camino} \rrbracket_T \cup \\ &\quad \llbracket \text{camino/camino} \rrbracket_T \cup \llbracket \text{camino/camino/camino} \rrbracket_T \cup \dots \end{aligned}$$

Ejemplo: $(\text{child}/?section/\text{child}/?section)^*$

La expresividad de Regular XPath

Nótese que Conditional XPath \subseteq Regular XPath:

`(paso/?test)*` es una fórmula en Regular XPath.

¿Qué tan expresivo es Regular XPath?

- ¿Es Conditional XPath = Regular XPath?
- ¿Con qué otro lenguaje lo podemos comparar?

Otro poco de historia: Autómata y lógica.

Indice

- Lenguajes de consulta para XML.
- Core XPath.
- Lógica de primer orden y XML.
- Conditional XPath.
- Regular XPath.
- Lógica de segundo orden monádica y XML.
- Monadic Datalog.
- Consultas con un número arbitrario de argumentos.

Autómata y lógica

¿En qué lógica podemos expresar expresiones regulares?

- Veamos primero el caso de las expresiones regulares sobre palabras.

Representamos una palabra $w = aabba$ sobre alfabeto $\{a, b\}$ como una estructura $\mathfrak{A}_w = \langle \{1, 2, 3, 4, 5\}, <, P_a = \{1, 2, 5\}, P_b = \{3, 4\} \rangle$.

Dada una expresión regular r , queremos construir una fórmula φ_r tal que para todo $w \in \{a, b\}^*$:

$$w \in L(r) \quad \text{si y sólo si} \quad \mathfrak{A}_w \models \varphi_r.$$

Autómata y lógica

FO no es suficiente: Considere $(aa)^*$.

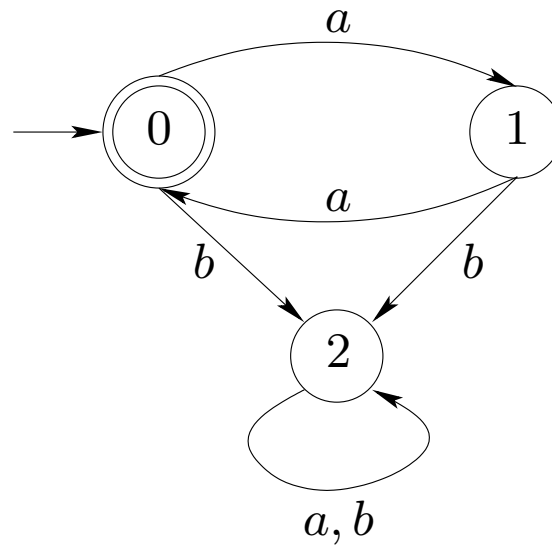
Hay una lógica que captura la noción de expresión regular: **Lógica de segundo orden monádica (MSO)**.

- Vamos a ver porque esto es cierto.

Autómata y MSO: Ejemplo

Dado: expresión regular $(aa)^*$ sobre alfabeto $\{a, b\}$.

El siguiente autómata acepta este lenguaje:



Autómata y MSO: Ejemplo

El autómata puede ser representado por una fórmula φ_r en MSO.

Para definir φ_r usamos las siguientes macros:

$$\mathit{primero}(x) \quad := \quad \neg \exists y (y < x)$$

$$\mathit{ultimo}(x) \quad := \quad \neg \exists y (x < y)$$

$$\mathit{sucesor}(x, y) \quad := \quad x < y \wedge \neg \exists z (x < z \wedge z < y)$$

Autómata y MSO: Ejemplo

$$\begin{aligned}\varphi_r = & \exists X_0 \exists X_1 \exists X_2 (\forall x (X_0(x) \vee X_1(x) \vee X_2(x)) \wedge \\ & \forall x (X_0(x) \rightarrow \neg X_1(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_1(x) \rightarrow \neg X_0(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_2(x) \rightarrow \neg X_0(x) \wedge \neg X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_a(x) \rightarrow X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_b(x) \rightarrow X_2(x)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_a(y) \rightarrow X_1(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_a(y) \rightarrow X_0(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_a(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x (\text{ultimo}(x) \rightarrow X_0(x))).\end{aligned}$$

Autómata y MSO: Ejemplo

$$\begin{aligned}\varphi_r = & \exists X_0 \exists X_1 \exists X_2 (\forall x (X_0(x) \vee X_1(x) \vee X_2(x)) \wedge \\ & \forall x (X_0(x) \rightarrow \neg X_1(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_1(x) \rightarrow \neg X_0(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_2(x) \rightarrow \neg X_0(x) \wedge \neg X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_a(x) \rightarrow X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_b(x) \rightarrow X_2(x)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_a(y) \rightarrow X_1(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_a(y) \rightarrow X_0(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_a(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x (\text{ultimo}(x) \rightarrow X_0(x))).\end{aligned}$$

Autómata y MSO: Ejemplo

$$\begin{aligned}\varphi_r = & \exists X_0 \exists X_1 \exists X_2 (\forall x (X_0(x) \vee X_1(x) \vee X_2(x)) \wedge \\ & \forall x (X_0(x) \rightarrow \neg X_1(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_1(x) \rightarrow \neg X_0(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_2(x) \rightarrow \neg X_0(x) \wedge \neg X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_a(x) \rightarrow X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_b(x) \rightarrow X_2(x)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_a(y) \rightarrow X_1(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_a(y) \rightarrow X_0(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_a(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x (\text{ultimo}(x) \rightarrow X_0(x))).\end{aligned}$$

Autómata y MSO: Ejemplo

$$\begin{aligned}\varphi_r = & \exists X_0 \exists X_1 \exists X_2 (\forall x (X_0(x) \vee X_1(x) \vee X_2(x)) \wedge \\ & \forall x (X_0(x) \rightarrow \neg X_1(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_1(x) \rightarrow \neg X_0(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_2(x) \rightarrow \neg X_0(x) \wedge \neg X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_a(x) \rightarrow X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_b(x) \rightarrow X_2(x)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_a(y) \rightarrow X_1(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_a(y) \rightarrow X_0(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_a(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x (\text{ultimo}(x) \rightarrow X_0(x))).\end{aligned}$$

Autómata y MSO: Ejemplo

$$\begin{aligned}\varphi_r = & \exists X_0 \exists X_1 \exists X_2 (\forall x (X_0(x) \vee X_1(x) \vee X_2(x)) \wedge \\ & \forall x (X_0(x) \rightarrow \neg X_1(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_1(x) \rightarrow \neg X_0(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_2(x) \rightarrow \neg X_0(x) \wedge \neg X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_a(x) \rightarrow X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_b(x) \rightarrow X_2(x)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_a(y) \rightarrow X_1(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_a(y) \rightarrow X_0(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_a(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x (\text{ultimo}(x) \rightarrow X_0(x))).\end{aligned}$$

Autómata y MSO: Ejemplo

$$\begin{aligned}\varphi_r = & \exists X_0 \exists X_1 \exists X_2 (\forall x (X_0(x) \vee X_1(x) \vee X_2(x)) \wedge \\ & \forall x (X_0(x) \rightarrow \neg X_1(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_1(x) \rightarrow \neg X_0(x) \wedge \neg X_2(x)) \wedge \\ & \forall x (X_2(x) \rightarrow \neg X_0(x) \wedge \neg X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_a(x) \rightarrow X_1(x)) \wedge \\ & \forall x (\text{primero}(x) \wedge P_b(x) \rightarrow X_2(x)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_a(y) \rightarrow X_1(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_0(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_a(y) \rightarrow X_0(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_1(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_a(y) \rightarrow X_2(y)) \wedge \\ & \forall x \forall y (\text{sucesor}(x, y) \wedge X_2(x) \wedge P_b(y) \rightarrow X_2(y)) \wedge \\ & \forall x (\text{ultimo}(x) \rightarrow X_0(x))).\end{aligned}$$

Expresiones regulares y MSO

Teorema (Büchi): Un lenguaje de palabras L es regular si y sólo si L es definible por una fórmula en MSO.

MSO es una buena alternativa para el caso de lenguajes regulares sobre palabras.

- ¿Qué pasa en el caso de documentos XML?

Teorema (Thatcher & Wright): Un lenguaje de árboles L es regular si y sólo si L es definible por una fórmula en MSO.

¡MSO es el nuevo objetivo!

- Nótese que es costoso evaluar una consulta en MSO.

Otra razón para usar MSO

Un **DTD (XML Schema)** especifica la estructura de los documentos usados en alguna aplicación:

$$\begin{aligned} db &\rightarrow book^* \\ book &\rightarrow author^*, section^* \\ author &\rightarrow \varepsilon \\ section &\rightarrow section^* \end{aligned}$$

¿Cómo podemos verificar si un documento T satisface un DTD D ?

- D define un lenguaje de árboles regular, por lo que existe φ_D en MSO tal que: **T satisface D si y sólo si $T \models \varphi_D$.**
- Basta con escribir φ_D en algún lenguaje eficiente que capture MSO.

¿Es suficiente con Regular XPath?

Nótese que Regular XPath $\not\subseteq$ Conditional XPath.

- $(\text{child}/?section/\text{child}/?section)^*$ es una consulta en Regular XPath que no puede ser expresada en Conditional XPath. e

¿Basta con Regular XPath para capturar MSO?

Teorema [GM05, BSSS06]: Hay una consulta en MSO que **no** es expresable en Regular XPath.

¿Existe un lenguaje eficiente que capture MSO?

Indice

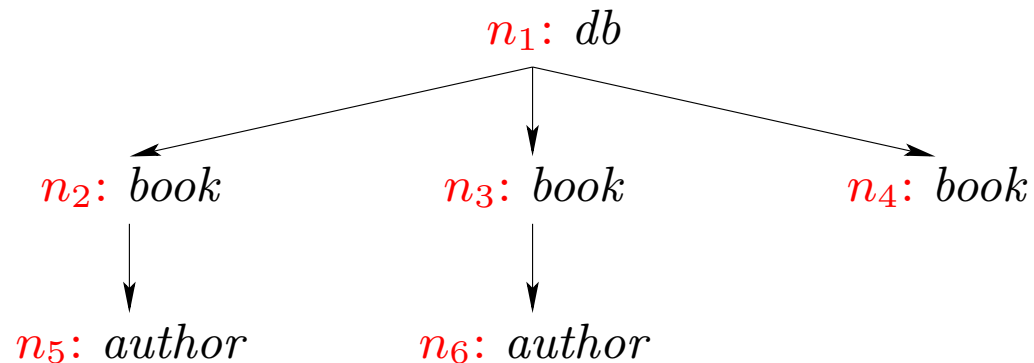
- Lenguajes de consulta para XML.
- Core XPath.
- Lógica de primer orden y XML.
- Conditional XPath.
- Regular XPath.
- Lógica de segundo orden monádica y XML.
- **Monadic Datalog.**
- Consultas con un número arbitrario de argumentos.

Monadic Datalog

Este lenguaje utiliza **reglas datalog** para definir un conjunto de nodos.

- A diferencia de XPath, este lenguaje no construye caminos, sino que extrae nodos que satisfacen alguna propiedad.

Dado árbol T :



Representamos T usando los predicados **extensionales**: `root`, `leaf`, `labeldb`, `labelbook`, `labelauthor`, `firstchild`, `nextsibling`, `lastsibling`.

Monadic Datalog

Donde:

$$\begin{aligned} \mathbf{root} &= \{n_1\} \\ \mathbf{leaf} &= \{n_4, n_5, n_6\} \\ \mathbf{label}_{db} &= \{n_1\} \\ \mathbf{label}_{book} &= \{n_2, n_3, n_4\} \\ \mathbf{label}_{author} &= \{n_5, n_6\} \\ \mathbf{firstchild} &= \{(n_1, n_2), (n_2, n_5), (n_3, n_6)\} \\ \mathbf{nextsibling} &= \{(n_2, n_3), (n_3, n_4)\} \\ \mathbf{lastsibling} &= \{n_4, n_5, n_6\} \end{aligned}$$

Monadic Datalog

Para definir una consulta usamos predicados **intensionales**.

- Estos predicados son **unarios**.

Ejemplo: La siguiente consulta extrae el conjunto de hijos de la raíz

$$Q(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$
$$Q(x) \leftarrow Q(y), \text{nextsibling}(y, x).$$

Monadic Datalog: Semántica

Semántica de Monadic Datalog: **Menor punto fijo.**

Para todo i :

$$Q_{i+1}(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$

$$Q_{i+1}(x) \leftarrow Q_i(y), \text{nextsibling}(y, x).$$

Entonces:

Monadic Datalog: Semántica

Semántica de Monadic Datalog: **Menor punto fijo.**

Para todo i :

$$Q_{i+1}(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$

$$Q_{i+1}(x) \leftarrow Q_i(y), \text{nextsibling}(y, x).$$

Entonces:

$$Q_0 = \emptyset$$

Monadic Datalog: Semántica

Semántica de Monadic Datalog: **Menor punto fijo.**

Para todo i :

$$Q_{i+1}(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$

$$Q_{i+1}(x) \leftarrow Q_i(y), \text{nextsibling}(y, x).$$

Entonces:

$$Q_0 = \emptyset$$

$$Q_1 = \{n_2\}$$

Monadic Datalog: Semántica

Semántica de Monadic Datalog: **Menor punto fijo.**

Para todo i :

$$Q_{i+1}(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$

$$Q_{i+1}(x) \leftarrow Q_i(y), \text{nextsibling}(y, x).$$

Entonces:

$$Q_0 = \emptyset$$

$$Q_1 = \{n_2\}$$

$$Q_2 = \{n_2, n_3\}$$

Monadic Datalog: Semántica

Semántica de Monadic Datalog: **Menor punto fijo.**

Para todo i :

$$Q_{i+1}(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$

$$Q_{i+1}(x) \leftarrow Q_i(y), \text{nextsibling}(y, x).$$

Entonces:

$$Q_0 = \emptyset$$

$$Q_1 = \{n_2\}$$

$$Q_2 = \{n_2, n_3\}$$

$$Q_3 = \{n_2, n_3, n_4\}$$

Monadic Datalog: Semántica

Semántica de Monadic Datalog: **Menor punto fijo.**

Para todo i :

$$Q_{i+1}(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$

$$Q_{i+1}(x) \leftarrow Q_i(y), \text{nextsibling}(y, x).$$

Entonces:

$$Q_0 = \emptyset$$

$$Q_1 = \{n_2\}$$

$$Q_2 = \{n_2, n_3\}$$

$$Q_3 = \{n_2, n_3, n_4\}$$

$$Q_4 = \{n_2, n_3, n_4\}$$

Monadic Datalog: Algunos ejemplo

Conjunto de libros que aparecen como hijos de la raíz:

$$Q(x) \leftarrow P(x), \text{label}_{book}(x).$$

$$P(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$$

$$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$$

Conjunto de libros que tienen al menos un autor:

$$Q(x) \leftarrow P_1(x), \text{label}_{book}(x).$$

$$P_1(x) \leftarrow \text{firstchild}(x, y), P_2(y).$$

$$P_2(x) \leftarrow \text{label}_{author}(x).$$

$$P_2(x) \leftarrow P_2(y), \text{nextsibling}(x, y).$$

Monadic Datalog: Complejidad

En general es costoso evaluar una consulta en Datalog .

- ¿Por qué queremos usar Monadic Datalog?

Teorema [GK04]: Es posible evaluar una consulta Q en Monadic Datalog sobre un árbol T en tiempo $O(|T| \cdot |Q|)$.

Veamos el algoritmo ...

Primer paso: Crear instancia ground

$Q(x) \leftarrow P(x), \text{label}_{book}(x).$

$P(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(x) \leftarrow P(x), \text{label}_{book}(x).$

$P(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_1) \leftarrow P(n_1), \text{label}_{book}(n_1).$
 $Q(n_2) \leftarrow P(n_2), \text{label}_{book}(n_2).$
 $Q(n_3) \leftarrow P(n_3), \text{label}_{book}(n_3).$
 $Q(n_4) \leftarrow P(n_4), \text{label}_{book}(n_4).$
 $Q(n_5) \leftarrow P(n_5), \text{label}_{book}(n_5).$
 $Q(n_6) \leftarrow P(n_6), \text{label}_{book}(n_6).$
 $P(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$
 $P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2), \text{label}_{book}(n_2).$
 $Q(n_3) \leftarrow P(n_3), \text{label}_{book}(n_3).$
 $Q(n_4) \leftarrow P(n_4), \text{label}_{book}(n_4).$
 $P(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$
 $P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(x) \leftarrow \text{root}(y), \text{firstchild}(y, x).$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(n_1) \leftarrow \text{root}(y), \text{firstchild}(y, n_1).$

$P(n_2) \leftarrow \text{root}(y), \text{firstchild}(y, n_2).$

$P(n_3) \leftarrow \text{root}(y), \text{firstchild}(y, n_3).$

$P(n_4) \leftarrow \text{root}(y), \text{firstchild}(y, n_4).$

$P(n_5) \leftarrow \text{root}(y), \text{firstchild}(y, n_5).$

$P(n_6) \leftarrow \text{root}(y), \text{firstchild}(y, n_6).$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(n_2) \leftarrow \text{root}(n_1), \text{firstchild}(n_1, n_2).$

$P(n_5) \leftarrow \text{root}(n_2), \text{firstchild}(n_2, n_5).$

$P(n_6) \leftarrow \text{root}(n_3), \text{firstchild}(n_3, n_6).$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(n_2) \leftarrow \text{root}(n_1), \text{firstchild}(n_1, n_2).$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(n_2) \leftarrow$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(n_2) \leftarrow$

$P(x) \leftarrow P(y), \text{nextsibling}(y, x).$

Primer paso: Crear instancia ground

$Q(n_2) \leftarrow P(n_2).$

$Q(n_3) \leftarrow P(n_3).$

$Q(n_4) \leftarrow P(n_4).$

$P(n_2) \leftarrow$

$P(n_1) \leftarrow P(y), \text{nextsibling}(y, n_1).$

$P(n_2) \leftarrow P(y), \text{nextsibling}(y, n_2).$

$P(n_3) \leftarrow P(y), \text{nextsibling}(y, n_3).$

$P(n_4) \leftarrow P(y), \text{nextsibling}(y, n_4).$

$P(n_5) \leftarrow P(y), \text{nextsibling}(y, n_5).$

$P(n_6) \leftarrow P(y), \text{nextsibling}(y, n_6).$

Primer paso: Crear instancia ground

$$Q(n_2) \leftarrow P(n_2).$$

$$Q(n_3) \leftarrow P(n_3).$$

$$Q(n_4) \leftarrow P(n_4).$$

$$P(n_2) \leftarrow$$

$$P(n_3) \leftarrow P(n_2), \text{nextsibling}(n_2, n_3).$$

$$P(n_4) \leftarrow P(n_3), \text{nextsibling}(n_3, n_4).$$

Primer paso: Crear instancia ground

$$Q(n_2) \leftarrow P(n_2).$$

$$Q(n_3) \leftarrow P(n_3).$$

$$Q(n_4) \leftarrow P(n_4).$$

$$P(n_2) \leftarrow$$

$$P(n_3) \leftarrow P(n_2).$$

$$P(n_4) \leftarrow P(n_3).$$

Primer paso: Crear instancia ground

$$Q(n_2) \leftarrow P(n_2).$$

$$Q(n_3) \leftarrow P(n_3).$$

$$Q(n_4) \leftarrow P(n_4).$$

$$P(n_2) \leftarrow$$

$$P(n_3) \leftarrow P(n_2).$$

$$P(n_4) \leftarrow P(n_3).$$

Segundo Paso: Calcular menor punto fijo

Ahora calculamos el menor punto fijo de la instancia ground.

Instancia ground:

- Tamaño: $O(|T| \cdot |D|)$.
- Tiempo para calcular menor punto fijo: $O(|T| \cdot |D|)$.

Monadic Datalog: Expresividad

Teorema [GK04]: Sobre árboles se tiene que $\text{MSO} = \text{Monadic Datalog}$.

¡Monadic Datalog es un buen lenguaje!

- Ha sido usado exitosamente en la practica [GKB⁺04].

Por fin tenemos lenguajes que capturan FO y MSO y que pueden ser implementados eficientemente.

- ¿o no?

Indice

- Lenguajes de consulta para XML.
- Core XPath.
- Lógica de primer orden y XML.
- Conditional XPath.
- Regular XPath.
- Lógica de segundo orden monádica y XML.
- Monadic Datalog.
- Consultas con un número arbitrario de argumentos.

Consultas con un número arbitrario de argumentos

Una consulta puede tener un número arbitrario de argumentos:

$Q(x, y, z)$: x e y son el primer y el último libro
escritos por z , respectivamente.

Dado un lenguaje \mathcal{L} , sea \mathcal{L}_n el conjunto de consultas en \mathcal{L} con n argumentos.

- Para cada n tenemos fragmentos FO_n y MSO_n .

Teoremas anteriores:

- Conditional XPath = FO_2 .
- Monadic Datalog = MSO_1 .

Consultas con un número arbitrario de argumentos

¿Podemos capturar FO_n y MSO_n ?

- No es claro como extender los resultados anteriores para capturar FO_n y MSO_n .

¿Qué podemos hacer para capturar FO_n y MSO_n ?

- No queremos generar lenguajes y probar que capturan el fragmento correspondiente.
- Queremos una metodología general.

Otros lenguajes

Los árboles son uno de los objetos más comunes en computación.

- Están presentes en muchas áreas de computación, en particular, en **ingeniería de software (verificación)**.

Sobre árboles se tiene que [BL05]:

- $L_\mu = \text{MSO}_0$ y $L_\mu^{\text{full}} = \text{MSO}_1$.
- alternation-free $L_\mu = \text{MSO}_0$ y alternation-free $L_\mu^{\text{full}} = \text{MSO}_1$.

¡Tenemos más alternativas!

- ¡Necesitamos una metodología general!

Capturando FO_n y MSO_n

Existe una manera general de construir lenguajes con buen poder expresivo y complejidad.

$\mathcal{I}^n(\mathcal{L}_1, \mathcal{L}_2)$: Lenguaje definido a partir de lenguajes \mathcal{L}_1 y \mathcal{L}_2 .

- Sus fórmulas tienen n argumentos.

Teorema:

- Si \mathcal{L}_1 captura FO_0 y \mathcal{L}_2 captura FO_1 , entonces $\mathcal{I}^n(\mathcal{L}_1, \mathcal{L}_2)$ captura FO_n (sobre árboles).
- Si \mathcal{L}_1 captura MSO_0 y \mathcal{L}_2 captura MSO_1 , entonces $\mathcal{I}^n(\mathcal{L}_1, \mathcal{L}_2)$ captura MSO_n (sobre árboles).

Capturando MSO_n : Ejemplo

Alternation-free $L_\mu = \text{MSO}_0$ y alternation-free $L_\mu^{\text{full}} = \text{MSO}_1$.

- Complejidad de ambos lenguajes es $O(|T| \cdot |Q|)$.

Corolario: $\mathcal{I}^n(\text{alternation-free } L_\mu, \text{alternation-free } L_\mu^{\text{full}})$ tiene el mismo poder expresivo de MSO_n .

Aún más: Complejidad de verificar si $T \models Q(a_1, \dots, a_n)$ es $O(|T| \cdot |Q|)$.

Trabajo futuro

- ¿Qué tan naturales son los lenguajes $\mathcal{I}^n(\mathcal{L}_1, \mathcal{L}_2)$?
- ¿Cuál es la complejidad de $\mathcal{I}^n(\mathcal{L}_1, \mathcal{L}_2)$ en aplicaciones reales?
 - Implementación.
 - ¿Pueden competir con los lenguajes existentes?

Referencias

- [BL05] Pablo Barceló and Leonid Libkin. Temporal logics over unranked trees. In *LICS*, pages 31–40, 2005.
- [BSSS06] Mikolaj Bojanczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. On the expressive power of pebble automata. In *ICALP*, 2006.
- [Cod72] E. F. Codd. Relational completeness of data base sublanguages. In: *R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California, 1972.*
- [GK04] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. *JACM*, 51(1):74–113, 2004.
- [GKB⁺04] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The lixto data extraction project - back and forth between theory and practice. In *PODS*, pages 1–12, 2004.
- [GKP05] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing xpath queries. *TODS*, 30(2):444–491, 2005.

- [GM05] Evan Goris and Maarten Marx. Looping caterpillars. In *LICS*, pages 51–60, 2005.
- [Mar04a] Maarten Marx. Conditional xpath, the first order complete xpath dialect. In *PODS*, pages 13–22, 2004.
- [Mar04b] Maarten Marx. Xpath with conditional axis relations. In *EDBT*, pages 477–494, 2004.
- [Mar05] Maarten Marx. First order paths in ordered trees. In *ICDT*, pages 114–128, 2005.